Technical Report GIT-GVU-98-32

The Design and Use of a Generic Context Server*

Daniel Salber and Gregory D. Abowd
GVU Center, College of Computing
Georgia Institute of Technology
801 Atlantic Drive, Atlanta, GA 30309
{salber, abowd}@cc.gatech.edu

Abstract

Although context awareness is a key component for perceptual user interfaces, we lack generic infrastructure for developing context aware applications. We propose a generic infrastructure based on context servers that store, share and archive contextual data. We describe a few applications we have built that take advantage of context sharing and context history. We then turn to the overall design of our context server and analyze in detail its services with a worked example.

1. Introduction

Context awareness is recognized as an important feature for ubiquitous and wearable computing. Context sensing and interpretation techniques are maturing and applications demonstrate the value of using context. However, bridging the gap between sensing and interpretation techniques on one hand and applications on the other hand is mostly done using ad hoc techniques. The lack of generic infrastructure requires developers to rely on custom solutions for handling context and hinders the development of new applications.

In this paper, we first look at common sources of context and emphasize the need for two often overlooked generic context handling features: context sharing in multi-user settings and context history. We describe applications that take advantage of these features. We then turn to the design of an infrastructure for supporting such applications. We first explain our goals and then describe our architecture for a generic context server. We finally explain its behavior in detail with the analysis of an example application that uses both shared context and context history to provide a group of users with notifications of their common web surfing interests.

1.1. What is Context?

Context is usually understood as information that the system can sense and process to facilitate human-computer interaction. In most cases, this information is peripheral to the user's task. Typically, context information is used to modify the system's behavior, trigger system actions, tag captured data or inform the user. Contextual information is acquired by specific sensors and is either posted to or polled by context-aware applications.

In the next section, we look at commonly used sources of context. Besides these, we identify two overlooked areas: sharing context in multi-user settings and exploiting context history.

1.2. Sources of Context

We distinguish four broad categories of context environments: physical, system, application, and social. The physical environment and notably the user's location is a most commonly used source of context [1, 4, 10, 12]. The system's environment, i.e., OS-level information, is a second popular source of context: nearby computing resources [13], network traffic [17] and connectivity [6] are presented to the user or used to tailor the interaction. Similarly, application environment data, such as the current text selection, provides context information that can help anticipate user actions [3, 5, 11]. Finally, the social environment is a relevant source of context information: information about people such as the presence of people, their identity, their activity, may be used by systems to either provide information to other users or tailor the system's behavior to a user's needs or preferences [1].

Although research has been carried out on identifying and tracking people in a scene, there are few convincing applications that take advantage of social context. Shared context information can provide richer social context and

* Information on the context server is available on the web at: http://www.cc.gatech.edu/fce/contextserver/

provide a foundation for applications. We also notice that context aware applications use context information at the present time. Context history also provides interesting information. We will discuss context sharing and history further in section 1.4.

1.3. Shared Context

There is previous work on sharing physical context information and namely location. For instance, a user's current location tracked by an active badge may not trigger any relevant actions for the wearer. But this information will help a colleague locate her to discuss an urgent problem or allow the secretary to forward a phone call [16]. However, context sharing can be extended profitably to a user's system and application context in order to provide social context to other users. Context sharing at the level of a group is another unexplored area.

While Daniel is editing this document, information related to his current task such as the name of his frontmost window or the paragraph he's working in are part of his application context information. Unless he uses a context aware application that for example exploits application context to anticipate his actions, it has little value outside his current task. However, for Gregory who is trying to decide if he should get to work on the paper or carry on an unrelated task, Daniel's application context provides him with social context that makes him aware of Daniel's current activity and can help him decide his course of action. Similarly, if Gregory is on the road and Word is not available in his system environment, Daniel's system should be made aware of it and be able to provide him with a format that Gregory can read.

Sharing context is also interesting for larger groups. An application we envision provides a group of people gathered in a social area with a display of news likely to interest most of them. By gathering each user's preferred daily Web sources of news, it decides to display on a large screen the news source that most people present prefer and the news items people haven't read yet. Another potential application relies on the inspection of people's unread email. When a member of a workgroup sends everybody else an urgent message (e.g., a meeting time change), the application allows the sender to check that everybody has read the message.

In group-level context sharing, context information from several users is gathered and synthesized into a new piece of context. Everybody being aware of the meeting time change is social context to the sender. This gathering of private information may appear like a potential privacy threat. To give users control over the information that is gathered, our infrastructure provides users with customizable privacy protection mechanisms (see section 4.2).

1.4. Context History

Most context aware applications deal with context data concerning the present. Except for context-based retrieval applications [7], there has not been much work done on the value of context data history. A marginal example is capture applications. In this case, context data is not remembered for itself, but because it is associated with some captured piece of data.

However, in everyday social relationships we rely naturally on historical data. For instance, when looking for somebody, we ask colleagues if they have seen this person in the recent past. This information may actually be more useful than the current location of the person. If she's been in her office, she saw the note left on her desk, or she's probably read her email. A user's recent context (her whereabouts) helps other users interact with her.

Similarly, a history of URLs visited may provide interesting clues to both the user and colleagues as to what tasks the user was engaged in.

Another example would be a context history-aware note-taking aid. It could look up meeting history information and pull up notes taken the last time the user was attending a meeting with the same persons. A similar application could be provided to students attending classes.

2. Applications

We have designed and built applications based on a generic context server. In this section, we describe three of them that exploit context sharing and context history capabilities of the infrastructure.

2.1. Where Have You Browsed Today?

The "Where Have You Browsed Today?" application aims at stimulating discussion between people who may share common interests based on their web surfing activity. In contrast to collaborative browsing tools [9], this application matches users' interests after they're done browsing to stimulate interaction when they may be more available to engage in discussion.

The application consists of a URLs logger that captures the current URL displayed in the user's web browser. Using the history feature, logs of visited URLs can be generated. At the end of the day, URLs logs are compared for common web pages or sites. The result is used to notify the users if they've been visiting the same pages or sites that day.

It is important to note that users do not know each other's URL logging history, which most users would consider private data. Only those URLs that are common to all users are revealed and only to them. Still, other options may be explored, like requesting permission from each user before sharing her common URLs.

2.2. Are You Reading Me?

The "Are You Reading Me?" application is intended to facilitate email communication. Suppose Daniel needs to send an urgent message to Gregory, who is usually overloaded with email. Email seems convenient but is it the right medium to get in touch effectively with Gregory today?

The "Are You Reading Me" application adds two functions to Daniel's email client:

- The first one allows Daniel to know how many messages are left unread in Gregory's Inbox.
- The second function allows Daniel to know how many previous messages from him to Gregory are still left unread.

Daniel's email client fetches these two pieces of information from Gregory's context. They allow Daniel to assess Gregory's current email load and the fitness of email for sending an urgent message.

Gregory has the possibility to restrict access to these pieces of his context. Typically one would want only close colleagues to be able to inquire about one's current email load.

2.3. Let's Have A Meeting!

The "Let's Have A Meeting!" application uses context to provide more efficient scheduling. When two people decide to have a meeting, they usually both create an entry into their schedule. Both entries have the same date, symmetrical information (A enters "meeting with B", B enters "meeting with A") and each user may add private notes. Using context may alleviate this duplication of work.

With our application, only one user has to create an entry in her schedule. She then shares this entry with the other person involved. If for instance, Gregory and Daniel decide to schedule a meeting, Gregory creates an entry labeled "meeting with Daniel" in his schedule. With an extra click, he shares this entry with Daniel. This action creates a symmetrical entry (i.e., "meeting with Gregory") in Daniel's schedule at the same date. If the meeting involves a third party, the name of the third party appears in both entries. Users can add personal information to the entry once it is created.

In this case, both Daniel's and Gregory's context information is used. When Gregory shares his meeting entry, his own context is queried for the user's name to reconstruct a complete meeting entry, namely to add Gregory's name as a participant to the meeting (this was implicit in Gregory's entry). Then, the complete meeting information is sent over to Daniel's scheduling application. This application in turn queries Daniel's context for the user's name and scans the meeting information to try to match the user's name. It then removes it if present and creates the entry in Daniel's schedule.

3. Context Infrastructure Design

In this section, we describe our design of a context infrastructure to support the applications described in the previous section. We first outline our design goals and observe that existing context infrastructures don't achieve them. We then turn to our overall design and architecture.

3.1. Design Goals

To provide the services required by the applications we just described, our infrastructure goals are threefold:

1) allow for networked applications to access local and remote context data in a heterogeneous environment;
2) accommodate a variety of applications, sensors, and operations on context data;
3) preserve the history of contextual data sensed.

With these objectives in mind, let us examine previous work on context infrastructures.

3.2. Previous Work

A few generic context-handling infrastructures have already been developed, notably Schilit's architecture for context aware mobile computing [13] and Hull *et al.*'s SitComp service [8].

Schilit's architecture mainly aims at storing context data in a repository accessible by networked applications running on mobile ParcTab devices or fixed computers. Applications as well as sensors manipulate context data directly and thus must be aware of the storage model. Independence of applications and sensors from context data is not guaranteed. Furthermore, no provision is made for storing the history of context data.

The SitComp (Situated Computing) service software component utilizes local sensors to provide situation information to applications through an API. Applications can either query the SitComp service or ask to be notified of context changes. SitComp also performs fusion of data from multiple local sensors and abstracts raw data into context information at a higher level of abstraction. SitComp puts the emphasis on these abstraction mechanisms and provides a clear separation between applications on one hand and context sensing and abstracting mechanisms on the other. However, SitComp is intended for applications running on a single computer and doesn't allow remote access. Although the authors envision using context history for context-based retrieval, SitComp does not seem to support this yet.

3.3. Global Design

Our infrastructure is comprised of context servers that maintain a dynamic model of context data. We first look at

the services provided by a context server and assess how this infrastructure achieves our objectives stated in section 3.1.

In our model, context data is sensed by devices and gathered in a repository by the computer these devices are attached to. Computers may be fixed or mobile and are connected through fixed or wireless networks. A computer is attached either to persons (individuals or groups) or places (e.g., rooms, buildings, vehicles). Each computer runs a context server that gathers raw local context data through sensors, stores it and provides context data access to local and remote applications. In addition, each context server runs services, called context synthesizers, that act on local or remote context data to generate context information at a higher level of abstraction.

Access to local and remote context data is provided through a context access API. This API guarantees independence of applications from sensors as well as from the particular storage model used. To allow access by heterogeneous clients, the API is a network API based on XML [15] and HTTP. Each context server runs an HTTP server as well as an XML parser. Requests and replies are encoded in XML. This mechanism is detailed in section 3.4.2. Similarly, a component mediates accesses from and to context sensors so they do not access the context repository directly. As in SitComp, two models of communication are supported: events and requests. In the events based mechanism, the context-generating component (either a sensor or the context repository) generates events to registered components when the sensed or stored data changes. In the request method, the context gathering component (either an application or the repository) polls the context-generating component when needed. This distinction reflects the dichotomy already observed in user interfaces between status, i.e., continuously available information and events, i.e., atomic, transient information [2].

As emphasized in SitComp, raw context data must sometimes be abstracted into higher level information. To achieve this, a context server hosts synthesizers. Synthesizers are pluggable modules that access context data through the API and generate new context data that is fed back to the context server. The plug-in mechanism will allow us to reuse or develop our own context abstraction components based on, e.g., heuristic rules or case-based reasoning. Examples of abstraction mechanisms include: deducing the state and country from a city name and assessing if a room is occupied or not by combining ambient lighting, sound level and presence sensors data. Synthesizers also aggregate context data from multiple context servers and perform comparisons as in the URLs log comparison example of paragraph 2.1. Another example of aggregation is the detection of spatial relationships (e.g., adjacency, inclusion) between geographical context information collected from multiple context servers.

Finally, to allow the use of context history, context servers log changes in context data and preserve historical data. They allow access to context data at any point in the past or retrieval of a value over a time interval.

3.4. Architectural Design

The context server's architecture is organized in three functional layers (see figure 1):

- The context management layer deals with context storage and acquisition;
- The context access layer provides an API for local and remote access to context data as well as access control mechanisms;
- Finally, the context-synthesizing layer provides abstraction mechanisms that act on local and remote context data e.g., for group context sharing.

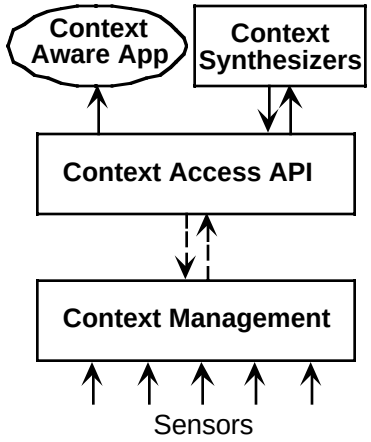

Figure 1. The overall architecture of the context server. The three rectangles constitute the context server. Arrows show context data flow between components. Dashed arrows denote XML encoded communications.

3.4.1. Context Management

The context management layer provides context storage and acquisition. It consists of three components: a persistent object database, a context acquisition component, and a context handlers component.

The repository of context data relies on an object database. Context data is organized hierarchically according to categories and is referred to according to a naming scheme. Top level categories correspond to entities context data is attached to, e.g., group, user, room, system, application. Lower levels partition each entity into categories of context data (e.g., sound, light and location for a room, relevant objects or properties for an application, etc.)

In our context data model, a number of common

attributes are attached to a piece of context data in addition to the current value. These attributes serve three functions:

- First, a timestamp attribute is used for historical purposes.
- Second, some attributes describe the data so that applications can make sense of it or request conversions e.g., units and reference systems for geographical coordinates,
- Third, other attributes give an estimate of the validity of the data. For instance, polled values have a lifetime attribute that may be queried by an application to check the data is still valid for its purpose.

In addition, the history of a given piece of data is stored in the database and can be queried. For data whose future values can be discovered (e.g., by looking up the user's schedule), the future scheduled values are also stored.

The context acquisition component provides a sensor-independent interface for storing and updating context data values. This component gets raw data from context sources and updates the database accordingly. This component insulates the context database from the sensors and guarantees independence with regard to the specifics of a particular sensor.

The context handlers component provides context data manipulation functions that mediate all accesses to the database. Its roles are twofold: it hides the actual structure of the data to guarantee independence with regard to the accessing components, including applications, and it provides access control capabilities. Remote access to any piece of context can be granted or forbidden by the user, or can be restricted to a list of authorized context servers. This privacy protection scheme however, places a burden on the user who has to configure access explicitly.

3.4.2. Context Access

The context access layer consists of the context accessors component. It provides an application interface to access context data remotely. The interface allows an application to get or set the value of a piece of context data and/or specific attributes. Context accessors query the context database through its context handlers component. The query is encoded as a remote procedure call (RPC) expressed in XML and routed via HTTP. A typical remote context query is shown in figure 2. The reply is also encoded in XML and is shown in figure 3.

In distributed heterogeneous environments typical of ubiquitous computing or mobile and wearable systems, the variety of platforms and programming languages in use makes interoperability a prime requirement. Interoperability at the networking level can be achieved by using TCP/IP. Our choice of HTTP as the transport protocol facilitates interoperability further: HTTP servers and client APIs are available on a large variety of platforms. Interoperability at the data format level is achieved by the use of XML to encode all exchanges over the network. Using XML, we can publish and share our context data hierarchy and naming scheme in a DTD (Document Type Definition) as well as the methods to access the context server, thus providing a public API to the context server. Furthermore, XML capabilities are available for a growing number of languages and platforms, making the port of our context access layer relatively easy.

```
<?XML VERSION="1.0"?>
<methodCall>
 <methodName>ContextServer.get</methodName>
 <params>
  <contextName>
   <contextCategory>
    <User/>
   </contextCategory>
   <contextItem>
    <EmailAddress/>
   </contextItem>
  </contextName>
</params>
</methodCall>
```

Figure 2. An XML/RPC query (simplified). An application calls the "get" method of a context server. It passes as a parameter the name of the requested piece of context: item EmailAddress in category User. This query is sent using HTTP POST.

```
<?XML VERSION="1.0"?>
<methodResponse>
 <params>
  <contextRecord>
   <contextValue>
    salber@cc.gatech.edu
   </contextValue>
  </contextRecord>
 </params>
</methodResponse>
```

Figure 3. The XML reply to the query of figure 2 (simplified). A context record is returned which contains the requested value. A context record may contain additional attributes for a piece of context (e.g., a timestamp).

3.4.3. Context Synthesizers

The context synthesizer layer is in charge of using raw context data to generate higher level of abstraction context data. It consists of two components: context abstractors abstract local context data; context aggregators generate higher-level context data from local and remote context servers.

The context abstractors component is in charge of extracting higher-level data from raw context data. It relies on an active values mechanism and recomputes high-level data whenever the raw data sources are updated. It stores its

result into the context database. For example, an application may need the current location of the user as a street name whereas only geographical coordinates are available. An abstractor would implement the required algorithms to generate geographical context as street names from the sensed geographical coordinates.

The context aggregators component consists of functions that operate on similar pieces of local and remote context data. They access local data through the context handlers interface and remote data through the context accessors. An example aggregator is the URL comparison function described in section 2.1 and detailed in section 4.2.

4. Context Servers at Work

In this section, we explain in detail how the context servers work. We revisit the "Where Have You Browsed Today?" application (WHYBT for short) described in section 2.1 and look at what is happening behind the scenes. This particular example takes advantage of most services of the context servers: history, context access and context sharing.

The current context server prototype is implemented in Frontier, a scripting language and environment that runs on MacOS and Windows [14]. Frontier provides us with a persistent object database, XML encoding and parsing, HTTP client and server support, and an AppleEvents or COM interface for inter-application communication.

From the point of view of the WHYBT application, things are pretty simple: it requests a comparison of the history of a piece of context data (the current URL) for all users concerned. It then sends an email to all users who have URLs in common. It thus needs to retrieve an additional piece of context information: the email address of the users involved. This application assumes that a context server is assigned to each user and is running on the user's workstation that she uses for browsing. More elaborate schemes could be devised to let users browse indifferently on a number of workstations or mobile devices.

The role of the context server for the WHYBT application is twofold: it first gathers and stores URL context data, and then interacts with the WHYBT application to serve its requests.

4.1. Gathering and Storing the Current URL

To allow handling by the context server of the "Current URL" piece of context, two components are needed: a sensor component that is able to grab the current URL from the browser application, and an entry in the context database. The sensor component is a script that connects through MacOS AppleEvents to the Netscape or Explorer browser of the user. At first, the script registers with the context handlers component (see 3.4.1) and declares it will provide "WebBrowser.CurrentURL" information as events. The context handlers then create an entry in the database if necessary. The sensor script then polls the value of the current URL at five-second intervals and generates an event to the context handlers whenever the value changes. In turn, the context handlers update the value stored in the database and maintain history information.

An interesting issue that arises is how long should history information be kept for a particular piece of context. Since the use of history information is up to the applications, we don't have an easy answer to this question. The current context server allows for arbitrary cutoff dates to reduce the amount of context data that is stored.

4.2. Interacting With the Application

A scheduler runs the WHYBT application every day late in the afternoon. The application is configured to serve a fixed list of users referred to by the name of their workstations. It runs on a group context server hosted by one of the users' workstations. It must first request the list of URLs common to all users for the current day and, if the list is not empty, request each user's email address from the context servers and notify each user by email.

The comparison of the URLs histories is performed by a dedicated "URLHistoryAggregator" script. It is a synthesizer and as such, registers with the local context handlers to declare the names of the context data that it takes as input and that it provides as output. In this case, the synthesizer uses "WebBrowser.CurrentURL" data and provides "WebBrowser.CurrentURLCommonSubset" data. It needs as an additional parameter the list of context servers that take part in the comparison as well as the history timespan to be considered. For aggregators, this list is usually provided by the client application.

Finally the WHYBT application queries local and remote context servers for the email addresses of the users involved. The XML query and reply used are shown in figure 2 and 3. It then generates email messages informing users of their common web visits for the day.

5. Conclusion

We have presented a generic context-handling infrastructure based on context servers. Context servers are particularly suited for sharing context data and providing access to context history. These features enable us to explore promising new applications. Our immediate goal is to develop more applications based on the context servers infrastructure. Of particular interest are applications that rely on elaborate context interpretation (e.g., from video images) and applications aimed at mobile users.

6. Acknowledgments

The first author is currently funded by a fellowship from INRIA whose support is gratefully acknowledged. We wish to thank members of the Future Computing Environments

group at Georgia Tech for fruitful discussions and particularly Anind Dey for insights and comments on our architecture and prototype applications.

7. References

[1] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper and M. Pinkerton. Cyberguide: A Mobile Context-Aware Tour Guide. *ACM Wireless Networks*, 3:421-433, 1997.

[2] G. D. Abowd and A. J. Dix. Integrating status and event phenomena in formal specifications of interactive systems. *ACM Software Engineering Notes*, 19(5):44-52, December 1994.

[3] Apple Research Laboratories. *Apple Data Detectors homepage.* http://www.research.apple.com/research/tech/AppleDataDetectors/, Apple Computer, 1997.

[4] N. Davies, K. Mitchell, K. Cheverst and G. Blair. Developing a Context Sensitive Tour Guide. *Proceedings of First Workshop on Human-Computer Interaction for Mobile Devices*, pp. 64-68, 1998, Glasgow, UK.

[5] A. Dey. Context-Aware Computing: The CyberDesk Project. *Proceedings of the 1998 Spring AAAI Symposium on Intelligent Environments*, 1998.

[6] M. R. Ebling and M. Satyanarayanan. On the Importance of Translucence for Mobile Computing. *Proceedings of First Workshop on Human-Computer Interaction for Mobile Devices*, pp. 69-72, 1998, Glasgow, UK.

[7] M. L. M. Flynn. Forget-me-not: Intimate computing in support of human memory. *Proceedings of FRIEND21: International Symposium on Next Generation Human Interfaces*, pp. 125-128, 1994.

[8] R. Hull, P. Neaves and J. Bedrod-Roberts. Towards Situated Computing. *Proceedings of IEEE ISWC'97, First International Symposium on Wearable Computers*1997, Cambridge, MA, USA.

[9] H. Lieberman, N. V. Dyke and A. Vivacqua. *Let's Browse: A Collaborative Web Browsing Agent*. http://lieber.www.media.mit.edu/people/lieber/Lieberary/Lets-Browse/Lets-Browse.html, MIT Media Lab, 1998.

[10] E. D. Mynatt, M. Back, R. Want and R. Frederick. Audio Aura: Light-Weight Audio Augmented Reality. *Proceedings of the ACM UIST'97 Symposium on User Interface Software and Technology*, p. 211-212, 1997.

[11] M. Pandit and S. Kalbag. The Selection Recognition Agent: Instant Access to Relevant Information and Operations. *Proceedings of Intelligent User Interfaces '97*, 1997.

[12] J. Pascoe, N. Ryan and D. Morse. Human-Computer-Giraffe Interaction: HCI in the Field. *Proceedings of First Workshop on Human-Computer Interaction for Mobile Devices*, pp. 48-57, 1998, Glasgow, UK.

[13] W. N. Schilit. *System architecture for context-aware mobile computing*. Ph.D. Thesis, 1995, Columbia University.

[14] UserLand Software. *Frontier 5.1*. http://www.scripting.com/frontier5/, UserLand Software, 1998.

[15] W3C XML Working Group. *Extensible Markup Language (XML) 1.0*. http://www.w3.org/TR/1998/REC-xml-19980210, World-Wide Web Consortium, 1998.

[16] R. Want, A. Hopper, V. Falcao and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91-102, 1992.

[17] M. Weiser and J. S. Brown. Designing Calm Technology. *Workshop on Ubiquitous Computing at CHI 1997*, 1997.